RC1/GB2004/002475

GB04/02475

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)

RECD 13 JUL 2004
WIPO PCT

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

I also certify that the application is now proceeding in the name as identified herein.

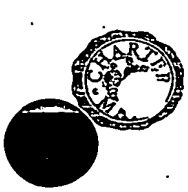
In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed

Dated 29 June 2004



GB 0313619.9

By virtue of a direction given under Section 30 of the Patents Act 1977, the application is proceeding in the name of:

SYMBIAN SOFTWARE LTD,
2-6 Boundary Row,
Southwark,
LONDON,
SE1 8HP,
United Kingdom

Incorporated in the United Kingdom,

[ADP No. 08843435001]

For Official use only

12 JUN 2003

THE PATENT OFFICE
K
12 JUN 2003
LONDON

13JUN03 E814638-1 D10092
P01/7700 0.00-0313619.9

Your reference **Dependency Trees (UK)**

The **Patent Office**

Request for grant of a Patent

Form 1/77 Patents Act 1977

1 Title of invention

A method of automatically analysing the structure of a software system

2. Applicant's details

0313619.9

[X] First or only applicant

2a If applying as a corporate body: Corporate Name

Symbian Limited

FILED 10/6/04

APPLICATION UNDER SECTION 30 (1977 ACT)

Country GB

07573132002

2b If applying as an individual or partnership

Surname

Forenames

2c Address

Sentinel House
16 Harcourt Street
London

UK Postcode W1H 1DS

Country GB

ADP Number

☐	Second applicant (if any)	
2d	Corporate Name	
	Country	
2e	Surname	
	Forenames	
2f	Address	
	UK Postcode	
	Country	
	ADP Number	
3	Address for service	
	Agent's Name	Origin Limited
	Agent's Address	52 Muswell Hill Road London
	Agent's postcode	N10 3JR
		07270457002
	Agent's ADP Number	C03274

4 Reference Number

Dependency Trees (UK)

5 Claiming an earlier application date

An earlier filing date is claimed:

Yes ☐ No ☒

Number of earlier application or patent number

Filing date

15 (4) (Divisional)	8(3)	12(6)	37(4)
☐	☐	☐	☐

6 Declaration of priority

Country of filing	Priority Application Number	Filing Date

7 Inventorship

The applicant(s) are the sole inventors/joint inventors

Yes ☐ No ☒

8 Checklist

		Continuation sheets	
Claims	1	Description	9
Abstract	1	Drawings	3+3 R2

Priority Documents	~~Yes~~/No
Translations of Priority Documents	~~Yes~~/No
Patents Form 7/77	~~Yes~~/No
Patents Form 9/77	Yes/~~No~~
Patents Form 10/77	~~Yes~~/No

9 Request

We request the grant of a patent on the basis of this application

Signed: Origin Limited) Date: 12 June 2003

(Origin Limited)

A METHOD OF AUTOMATICALLY ANALYSING THE STRUCTURE OF A SOFTWARE SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to a method of automatically analysing the structure of a software system, such as an operating system for a computing device.

2. Description of the Prior Art

When trying to gain a high-level view of the inter-dependencies between the many executables (perhaps 500 or more) in an operating system, a view manually arrived at even by a skilled analyst quickly gets obscured by the sheer number of relationships. Hence, it is very difficult to identify inappropriate coupling between components of the OS (e.g. a component where one of its executables depends on a high-level other component for no good reason, indicating perhaps bad layering or inappropriate inclusion of an executable in the component).

Further, it is very helpful to be able to calculate the order in which executables and groups of strongly inter-dependent executables (e.g. components) should be built to ensure that executables with the least number of dependenceis are built first. But this is again difficult, even for the skilled analyst, and can take several days. Performing regular (e.g. daily or weekly) re-calculations as an OS build progresses is therefore impractical when relying on a highly skilled, but essentially manual process.

Glossary

Term	Description
Dependency	An executable is said to depend on another executable if it calls one or more of the exported functions in the other executable.
Executable or executable file	Generic term used to specify either a DLL or EXE, containing binary code directly runnable by the computer. A DLL provides exported functions for use by other executables. An EXE is a self-contained

	program and provides a single entry point.
Exported functions	The set of functions provided by a DLL that may be called by other executables.

SUMMARY OF THE INVENTION

The invention automatically produces a structural analysis of a software system's executables, separated into levels based on the concept of 'dependency depth'.

Given a simple list of executables' dependencies, a tool that implements the invention automatically produces a dependency table sorted by 'dependency depth' level, with the least dependent executables listed at the bottom and with the most dependent at the top. Executables with circular dependencies are not problematic, with the executables involved automatically being treated as being at the same level as each other.

The tool achieves this by assigning a unique and well-defined 'dependency depth' number to each executable. This number defines how many levels exist in the executable's dependency tree. This number may be calculated by expanding that executable's dependency tree recursively so that each executable is listed in expanded form exactly once in the tree for the right-most occurrence only, and is listed in collapsed form for all other occurrences. This guarantees that the tree is as deep as possible and is therefore also unique, making it usable for sorting a set of executables according to their dependency depth numbers.

Using the table that is produced in this way simplifies the production of a block diagram based on dependency, with executables at the same dependency level grouped together horizontally in the block diagram. Hence, the present invention provides a mechanism that organises the executables in a rational and repeatable manner that clarifies the high-level view of the inter-dependencies between the many executables. It can also be used to decide the order in which executables need to be built, where the least dependent executable is built first.

With further information giving the grouping of executables into components, the same technique may be used to find the dependency depth number of a component, where a component is a group of related executables which have strong inter-dependencies, usually built and deployed as a unit. Component M depends on component N if any executable in M calls a function of any executable in N.

Summary of the benefits of the present invention

- Better understanding of OS interdependencies through a systematic, reliable and comprehensive analysis of the system architecture. A system architect can find inappropriate coupling between components of the OS – e.g. a component where one of its executables depends on a high-level other component for no good reason, indicating perhaps bad layering or inappropriate inclusion of an executable in the component;

- Enables automatic, rapid and reliable calculation of the order in which components should be built. Items at low levels are guaranteed to be buildable without previously building items at higher levels. Circular dependencies need to be built together;

- Results of the analysis can be used by other tools;

- Leads to improved modularity, aiding rollout of independent features;

- Helps produce a block diagram of the OS.

DETAILED DESCRIPTION

To simplify this description, we will use specific examples of very simple hypothetical Operating Systems that have only a small number of executables.

If executable A calls a function in executable B and another function in executable C, A is said to depend directly on both B and C. This can be represented by the following line:

A: B C

where the executable on the left of the colon depends directly on the executables on the right.

Example 1: No circular dependencies

The following table specifies the complete direct dependency structure of a hypothetical OS with six executables, A, B, C, D, E and F:

A: B C

B: C

C:

D: A C E

E: A

F: E

Using this direct dependency structure, a dependency tree can be generated for each executable which includes direct dependencies as well as their dependencies and so on recursively, as shown in **Figure 1**:

In these representations of the dependency trees, a direct dependency is indented by one tab to the right of the executable that depends on it, so as before:

1. E depends directly on A only
2. D depends directly on A, C and E
3. A depends directly on B and C
4. B depends directly on C only
5. C depends on nothing.

This can be simplified by collapsing sub-trees that are repeated in the tree, giving the following trees, where a '+' indicates a collapsed executable sub-tree, expanded *further to the right* somewhere else in the tree, as shown in **Figure 2:**

Collapsing repeats is important for the tool's memory and speed efficiency when analysing a real OS, with potentially thousands of executables and millions of repeated sub-trees within each tree. See an algorithm for achieving this efficiently below.

Each executable can then be assigned a unique *dependency depth number* by counting the levels of indentation, given by the maximum number of dots in any row for the specified executable's tree above.

Executable	Dependency Depth Number
A	2
B	1
C	0
D	4
E	3
F	4

The dependency depth number can now be used to partition the OS into levels with executables having the lowest dependency depth number at the bottom as follows

Level 4:	D, F
Level 3:	E
Level 2:	A
Level 1:	B
Level 0:	C

Example 2: Includes circular dependencies

The following table specifies the complete direct dependency structure of a second hypothetical OS:

A: B C

B: C

C: A

Using this direct dependency structure, the dependency trees are represented as follows – where recursion stops on reaching a circular dependency to avoid infinite regress, as shown in **Figure 3:**

Again the unique *dependency depth number* is found by counting the levels of indentation, given by the maximum number of dots in any row above.

Executable	Dependency Depth Number
A	3
B	3
C	3

Partitioning the OS into levels again using these dependency depths produces the following:

Level 3:

A, B,C

Note that the circular dependencies cause empty levels 0, 1 and 2.

Efficient algorithm for collapsing repeated sub-trees

A real OS has potentially thousands of executables and millions of repeated sub-trees within each tree, so an algorithm for collapsing repeats efficiently and in an easily searchable and parseable way, is very important for a workable tool.

As described below, the finally generated tree for D from example 1 above can be stored efficiently as a single easily computer-searchable and parseable string as follows:

D's tree = ' A+ C+ E:1{ A:2{ B:3{ C:4{}C }B C+ }A }E '

The format of a collapsed executable Y is simply 'Y+ '

The format of an executable Z that has a circular dependency on it is 'Z+(circular) '

The start tag for executable X's expansion at indentation level L is

'X:L {'

and its end tag is

'}X'

and between the braces are the details for the executables X depends on which is empty for an executable with no dependencies.

To build e.g. D's tree from example 1, named D-tree here for convenience, follow these steps, noting that substrings enclosed by angle brackets represent variable quatities:

1. Initialise D-tree to empty string ""
2. For each executable used by D (i.e. for X=A, X=C and X=E) do the expansion in step 3 at level L=1
3. Add used executable X at level L:

a) If X equals D, add 'X+(circular) ' and finished step 3 for X

b) Search for previously added *partial* expansion 'X:M {'in D-tree with no terminating '}X' and if found, signifies a partially built expansion and therefore a circular dependency, so add 'X+(circular) '

c) Search for previous expansion 'X:M {<anyText>}X ' in D-tree where M is a previously added level number

d) If found and L is less than or equal to M, add ' X:L+' and finished step 3 for X

e) If found and L is greater than M, replace previously added 'X:M {<anyText>}X' by 'X:M+'

f) Now add the expansion

i. Add 'X:L {' marking expansion start for X

ii. Add expansion for each executable used by X at level L+1 (i.e. repeat step 3 for all executables used by X recursively)

iii. Add '}X ' marking expansion end for X

Note that at step 3f) above the previously found expansion from step 3a) above can't be used for further efficiency, because that expansion will include executables that themselves are expanded to a different level than required in step 3f) above.

Here is the full tree expansion for the OS described in example 1:

```
A => 'B:1{ C:2{}C }B  C+ '
B => 'C:1{}C '
C => ' '
D => ' A+  C+  E:1{ A:2{ B:3{ C:4{}C }B  C+ }A }E '
E => 'A:1{ B:2{ C:3{}C }B  C+ }A '
F => 'E:1{ A:2{ B:3{ C:4{}C }B  C+ }A }E '
```

And here is the expansion for the OS described in example 2:

```
A => 'B:1{ C:2{ A+(circular) }C }B  C+'
B => 'C:1{ A:2{ B+(circular)  C+(circular) }A }C'
C => 'A:1{ B:2{ C+(circular) }B  C+(circular) }A'
```

The maximum number in this string gives the dependency depth for the executable.

Symbian OS v7.0s with more than 550 executables produces a full definition of this kind that has size 810K.

CLAIMS

1. A method of automatically analysing the structure of a software system, comprising the step of using an automated software tool to determine the dependency depth level of each of several executables and to then partition the system by organising the executables into their respective dependency depth levels.

2. The method of Claim 1 in which the tool outputs a dependency table in which each of the executables is sorted according to dependency depth.

3. The method of Claim 2 in which executables with circular dependencies are placed at the same level.

4. The method of Claim 3 in which the tool assigns a dependency depth number to each executable, calculated by expanding each executable's dependency tree recursively so that each executable is listed in expanded form exactly once in the tree for the right-most occurrence only, and is listed in collapsed form for all other occurrences.

5. The method of Claim 4 in which the tool is further able to determine the dependency depth level of each of several components, each comprising a group of related executables with strong inter-dependencies.

6. The method of Claim 1 in which the software system is an operating system.

7. A software based tool that automatically analyses the structure of a software system, the tool programmed to determine the dependency depth level of each of several executables and to then partition the system by organising the executables into their respective dependency depth levels.

8. An operating system which is automatically analysed during its design, implementation or maintenance phases by an automated software tool that determines the dependency depth level of each of several executables and then partitions the system by organising the executables into their respective dependency depth levels.

ABSTRACT

A METHOD OF AUTOMATICALLY ANALYSING THE STRUCTURE OF A SOFTWARE SYSTEM

The invention automatically produces a structural analysis of a software system's executables, separated into levels based on 'dependency depth'. Given a simple list of executables' dependencies, the tool automatically produces a dependency table sorted by level, with the least dependent executables listed at the bottom and with the most dependent at the top. This organises the executables in a rational and repeatable manner that clarifies the high-level view of the inter-dependencies between the many executables. It can also be used to decide the order in which executables need to be built where the least dependent executable is built first.

Figure 1

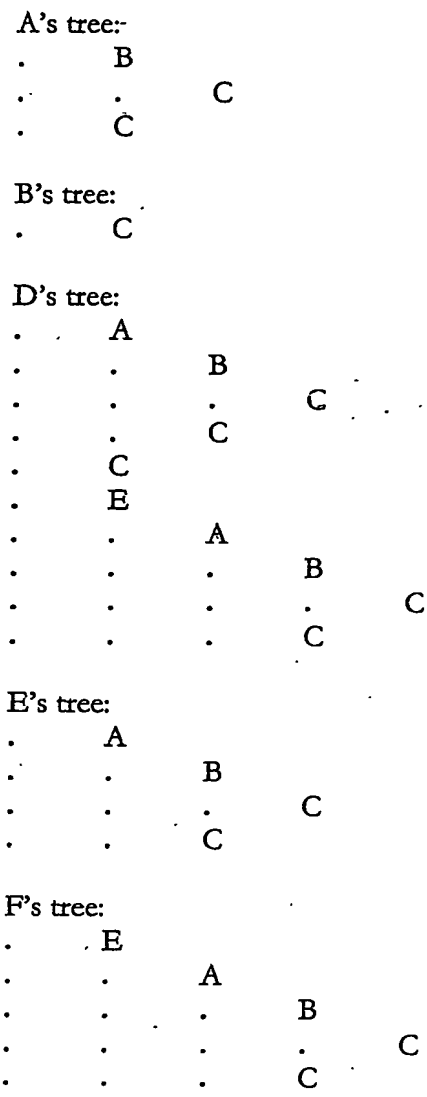

A's tree:-
B
C
C
B's tree:
C
D's tree:
A
B
C
C
C
E
A
B
C
C
E's tree:
A
B
C
C
F's tree:
E
A
B
C
C

Figure 2

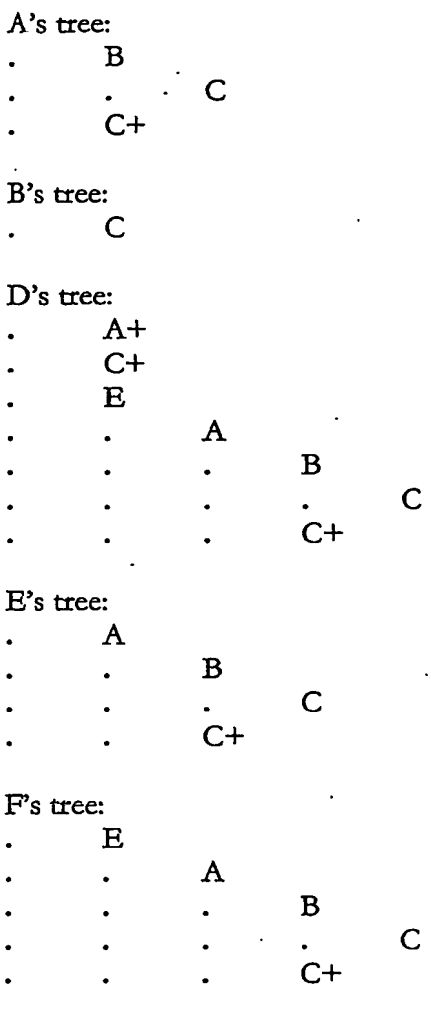

A's tree:
B
C
C+
B's tree:
C
D's tree:
A+
C+
E
A
B
C
C+
E's tree:
A
B
C
C+
F's tree:
E
A
B
C
C+

Figure 3

A's tree:

```
.       B
.       .       C
.       .       .       A+ (circular)
.       C+
```

B's tree:

```
.       C
.       .       A
.       .       .       B+ (circular)
.       .       .       C+ (circular)
```

C's tree:

```
.       A
.       .       B
.       .       .       C+ (circular)
.       .       C+ (circular)
```

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.